Experience Using a Project-Based Approach in an Introductory Programming Course

David Davenport

***Abstract*—This paper describes how and why our department chose to integrate a sophisticated visual programming environment (DELPHI) into the introductory programming course (CS101/2) of the first-year undergraduate engineering curriculum. It also reports on the outcome of this venture, which involved a switch from a conventional lecture-based approach to teaching CS102 to a project-based approach. While it meant more work for both students and teachers the overall response from both parties was very positive. Indeed, the resulting designs and subsequent survey results seem to confirm that students achieved a better appreciation and understanding of the course's objectives.**

***Index Terms*—Computer programming languages, CS1, CS2, DELPHI, introductory programming, PASCAL, project-based learning, ROBO, software engineering education, student centered learning, undergraduate engineering education.**

I. Introduction

A PROGRAMMING class in the first-year undergraduate curriculum is something of an enigma. It seems somehow out of place alongside conventional subjects, such as physics and calculus. While most students find it useful and even enjoyable to learn how to use a computer, teachers often find it difficult to arrange and justify the course's contents. Somehow it is necessary to balance the need to give students general computer literacy and vocational skills, with the demands for a rigorous engineering education. The result of these often-conflicting objectives is wide variations in style, a constant reappraisal of contents, and never-ending arguments over which programming language to use. Rapid advances in technology merely serve to aggravate the situation and ensure that introductory programming classes, hereafter referred to as CS101/102, are in a continual state of flux.

This paper describes our experience in designing and teaching this course. It begins by examining the rationale for including programming in the first-year curriculum and derives a clear set of objectives that it is felt such a course should be designed to achieve. It then describes the particular implementation, which takes students with no prior experience of computers and, via a combination of lectures, laboratory assignments, and group-based project work, enables them to produce sophisticated software products. The survey results, presented in the final section, evaluate the strengths and weaknesses of this approach.

Manuscript received May 8, 1997; revised August 21, 2000.

The author is with the Computer Engineering and Information Science Department, Bilkent University, Ankara, Turkey (e-mail: david@bilkent.edu.tr).

Publisher Item Identifier S 0018-9359(00)10083-4.

II. Why Teach Programming in the First Year?

Computer science (CS) has not traditionally been thought of as science in the same mold as physics, chemistry, and biology. It is certainly not a science of the natural world, but, some would argue, of the artificial. Thus, CS is taught in first-year undergraduate courses, not so much because it is part of the world around us, as because it is a tool with which to help explain and analyze it, much as calculus is (and philosophy was?)

Programming entered the curriculum only a few decades ago, as computers first moved out of research labs and into more general use. At that time, anyone who wanted to make use of the machine had to be able to write programs. It was also necessary to know how to punch cards and to be extremely patient while waiting, sometimes days, for the next output, after correcting the inevitable missing comma error which had ruined the previous day's run! In that era it made sense to teach budding engineers about the computer so that they could make good use of it later. Today, however, the computer is all pervasive. Moreover, there are now cheap, readily available, package programs that can accomplish nearly all the basic tasks students' face. There is almost no need to write programs anymore!

Why then, are undergraduates still taught to program? In the case of computer science students the rationale is perhaps obvious, however, CS101/2 tends to be taught to all freshmen engineers. Why? Probably because university curricula have not yet caught up with reality. Universities tend to be rather conservative. This is not necessarily a bad thing, of course, however, it does leave the (interim) problem of justifying what is taught (or melding it to other ends?) Indeed, this is all the more difficult given that, in many countries, students now entering university already have a good background in computing.

The response to these changes has been an attempt to justify the course on the basis that it enhances logical thinking and problem solving skills. While this may be true the evidence seems far from clear and, besides, there may be simpler/easier ways of teaching them. In any case, those who enter university (particularly engineering departments) already tend to have a reasonable (albeit, perhaps intuitive) grasp of such skills otherwise they would never have gotten there in the first place.[1]

[1]This may vary somewhat across countries and cultures. In Turkey, a large youth population and the perceived need to obtain a good education, has resulted in a fiercely competitive nationwide university entrance examination system. The huge demand for an engineering education means that such departments are able to select the very best students—those that have excelled in math and science at high school, and are therefore much better than average, at least in terms of their ability to think logically and to solve certain types of problems.

III. What Should be Taught?

How then, might the teaching of an introductory programming and algorithms course be justified in today's first-year undergraduate curricula? What should be taught? Five overreaching objectives come to mind. Students should be encouraged to develop the following:

1) an engineering approach to problem solving and design;
2) improved communication skills;
3) the ability to work together in a team;
4) the ability to use complex modern computer systems;
5) the ability to learn independently.

The first recognizes that, while university students are relatively good at problem solving and are generally capable of clear logical thinking, their abilities tend to be very intuitive. That is, they have a natural (instinctive) mental capacity that makes them particularly adept at answering exam questions. Unfortunately, such skills do not necessarily translate into success in the larger engineering arena. The difficulty is one of scale. People can often handle small problems because they can readily hold the entire task in their head and apply their intuition to solve it. Nowadays, however, practicing engineers have to deal with very complex systems. In such cases, there is simply no way to apply intuition since the system is beyond the individual's ability to fully comprehend. The aim then, should be the teaching of a methodology for tackling large engineering type problems [1]. The mentality, common in programming circles, which relies on experiment and test, must be replaced by a rigorous design methodology. It is important to cultivate the idea that engineering is about producing complex products that function "right-first-time and are maintainable."

One of the difficulties faced here is that CS101/2 is an introductory course, so most students are grappling with very basic programming concepts. It is thus almost impossible to assign them large problems, since they plainly lack the means to solve them. Yet if they do not get to work on complex problems they do not appreciate the need for a methodology in the first place. This dilemma would appear to be at heart of most of the upheaval experienced in CS101/2.

The second objective should be to help students develop their communication skills, both written and oral. Engineering students have traditionally viewed language skills as irrelevant or, at the very least, secondary to technical ability. They resist writing reports, considering it just boring overhead, not part of the real intellectual work! But this attitude is regrettable for two reasons. First, and perhaps foremost, the very process of design requires and is significantly aided by putting one's ideas down on paper. Not only is this necessitated by personal limited cognitive abilities, but it is usually the case that the very act of communicating forces one to better organize and clarify his/her ideas. Moreover, it opens up those ideas to criticism by others. The result, then, is a much more focused view of the problem and less chance of overlooking something. The second reason why neglecting communication skills is regrettable, is that, to be successful in today's highly competitive markets, engineers really do have to be extremely good communicators. They must be able to talk with customers, first to understand their requirements, to negotiate and agree upon a product specification, and later to "sell" the resulting design back to them. And, of course, engineering is no longer a solitary discipline. Building complex products involves many people, often with a variety of different skills and backgrounds, so it is vital to the success of a project that everyone be able to communicate effectively with other members of the team. This brings us to the third major objective that CS101/2 should aim for. Given that engineering is now very much a team effort and that people do not necessarily find it easy to work together, it makes good sense to help students develop these all-important interpersonal skills as early as possible. Replacing the highly competitive attitude prevalent in the high school system[2] with one of cooperation and sharing can only be good for everyone.

Finally, the fourth and fifth objectives are intended to get students familiar with the use of sophisticated modern computer systems. This entails not just basic computer literacy, the use of word processors and communications, but also complex programming environments. Early familiarity with such systems will make it easier for students to learn to use other products and even to design new software themselves. Initially, of course, students need to be taught to use certain programs, however, it is important that students acquire the capability to learn on their own. This is vital in the computer industry where new products appear very frequently. Giving students the confidence that they can find out about, and learn to use complex systems, from manuals, books, online help, the Internet and friends, is extremely important and one of the major reasons for the shift toward student-centered learning.

IV. Course Design Considerations

Having decided upon what would appear to be justifiable objectives for the course, the next step is to determine how to put them into practice. Obviously, the students, the way the course has been implemented and taught in the past, the overall curriculum, and the available equipment, will have a significant influence on decisions at this stage. Each of these aspects will now be examined in more detail.

A. Students and Equipment

Bilkent University is a private institution, the first in Turkey. Set up as a center of excellence in research, it has consistently managed to attract the very best students. Every year more than a million students apply to enter Turkish universities. From the national entrance exam Bilkent accepts and awards scholarships to, only about 150 students from those in the top three hundred (or less) for admission to the engineering faculty. The students are thus very bright, competitive, and highly self-motivated. Unlike their counterparts in more developed countries, however, Turkish students generally have little experience with computers. Although the situation is changing as computers gradually find their way into high schools and homes, currently at least 50% of freshmen students have absolutely no prior exposure to computers. For this reason, Bilkent has placed considerable emphasis on making computer facilities available to

[2]The situation will obviously vary widely from one country to another. Circumstances in Turkey have resulted in a highly competitive high school education system (see footnote [1].)

both students and staff. While the backbone of the university's network is formed of Unix machines, student labs have always been equipped with PC compatibles, originally XT's running MSDOS, nowadays, Pentiums running MS Windows NT.

B. The Existing Course

The introductory CS101/CS102 programming and algorithms course is mandatory for all freshmen engineering students entering Bilkent University. Since the university's inception ten short years ago, the course has been taught using Turbo Pascal and Koffman's textbook [2]. Both the software and the book have gone through many editions, however, the basic content and form of the course has remained unchanged. The first semester gets students familiar with using the computer and provides an introduction to basic programming concepts. These include the notions of comments, sequence, decision, repetition, variables and constants, procedures and functions, parameters, arrays and records. Emphasis is placed on top-down structured algorithm development as a means of problem solving. The second semester continues with more sophisticated programming techniques, stressing the need for time and space efficient solutions to real-world problems. File processing, recursion, searching, and sorting are covered, including binary search, hashing, selection sort, quicksort and mergesort, plus a variety of data structures such as lists, stacks, queues and trees. This corresponds broadly with the ACM recommendations for CS1/2 courses [3].

A midterm exam (25%), a final exam (30%), weekly laboratory assignments (30%), and homework, usually essays on social or other aspects of computing (15%), determine grades. In recent years, students have been given the option of undertaking an individual project in place of the homework component in the second semester only. This has proved successful in terms of motivating students, but has failed to meet broader objectives (such as those outlined above) due mainly, perhaps, to a lack of rigorous reporting requirements, necessitated by its low overall weight in the course.

One departure from what would otherwise be a very conventional approach to teaching CS101/2, occurs during the first few weeks of the course. A tool, called Robo [4], developed specifically to acquaint students having absolutely no prior background in the subject, with the basic concepts and rationale of programming and software engineering, is employed. In essence, Robo is a very simple stand-alone implementation of LOGO's turtle graphics [5], [6]. It provides a very concrete means of introducing basic ideas and skills, including sequence, repetition, comments, meaningfully named procedures and parameters, top-down structured design, pre/post conditions, etc. This program and an outline course are freely available to teachers [7].

In Bilkent, CS101/2 was followed in the second year by a one-semester course on data structures and a two-semester sequence on fundamental structures of computer science. These courses assume that students have gained the rudiments of programming in the first year and hence concentrate on more analytical skills. They also switch to the C language. Several common complaints have echoed down the years, 1) there is too much repetition of data structures; 2) students do not have enough familiarity with C; and 3) students lack design skills. It was hoped to resolve some of these problems by reducing the data structures component in CS102 and concentrating instead on engineering design skills.

C. The Dreaded Question of Language

As always, one of the major choices to be made concerns the programming language to be taught [8]. While Borland's Turbo Pascal has been used for the last ten years, there has been significant and recurring pressure to change to C, C++ and even Scheme (not to mention other possibilities such as ADA, Oberon, Eiffel and so on!) The reasons usually given for considering such alternatives are 1) they are cleaner, purer, academically more acceptable or 2) they are in wide use and hence more commercially acceptable. Both of these are true, which is exactly why we chose to stay with Pascal!

Pascal seems to offer a good compromise. Of course, standard (ANSI) Pascal is too restricted to be useful for commercial applications (and hence education), however, almost all actual implementations offer a full range of extensions. Although obviously not as portable or widely used as C, Pascal is nevertheless very popular. Nowadays, C's portability is less significant, because the large user-interface component typical of windows-based programs is not directly transferable across operating systems. Compared to C, Pascal has a relatively natural English-like syntax making it easier for students to learn, allowing them to concentrate on program design, which presumably is what is important! Although Pascal is certainly not the purist's language, it is very concrete which also makes it a good choice for newcomers, and the fact that it is a conventional language means that having once mastered Pascal, students can very quickly pick up most commercial languages. In one of its latest guises, (Borland Pascal version 7.0) it also provides a relatively smooth transition to object-oriented-programming. Last, but certainly not least, Pascal is now available in a powerful integrated visual programming environment, in the guise of Borland's Delphi [9]. This product allows students to be introduced to complex event-driven object-based systems with GUIs, and, as will be seen shortly, have them producing very good results extremely quickly. Delphi not only offers a superbly clean implementation of object-oriented features, it also handles much of the mundane program writing, freeing students to concentrate on more important matters.

V. The Final Course Design

Given that up to 50% of freshmen students entering Bilkent have absolutely no experience with computers, it was decided that the first semester CS101 course should retain its conventional lecture/lab framework. Since these students are usually away from home for the first time, in a new environment, making new friends, etc., this option would seem to offer the least additional stress and, anyway, would ensure that they did learn the necessary concepts. It also gives the opportunity to reorient those students who have learned programming in school, using the Basic language! It is these students, who tend to program by trial and error, who generally have the most difficulty.

So CS101 stayed pretty much the same as described above, although an additional lab, specifically intended to get students familiar with word processors and network communications in the form of MSWord and Netscape, was introduced. CS102, however, underwent a complete redesign. In line with the objectives

outlined previously, a team-based project approach was chosen. In order to smooth the transition and to provide additional techniques which students might need in their projects, some lectures continued to be given as before. This allowed the teaching of certain important topics, such as recursion, direct-access file processing and some pointer-based data structures (list, stacks, and queues) although at a much reduced level. Most such lectures were given early in the semester, others being slotted in when project work allowed.

The projects themselves were much the more important component of the course. They accounted for 70% of the overall grade, a conventional final exam and laboratory assignments making up the remaining 30%. Students were allowed to form their own groups and to select their own project, choosing either from a list of possible projects or proposing one of their own (subject to the instructors approval.) It was decided to enforce the figure of six students per group in order to limit each class to a maximum of eight groups, which was considered to be the most an instructor could reasonably be expected to follow. While there was a danger that this might allow some students to avoid work, it would provide enough hands/minds to tackle sensible sized projects assuming they did work. Having relatively large groups also has the advantage (from the instructor's point of view) of making it more likely that group-related problems will arise, forcing students to experience and learn about real teamwork!

Projects were divided up into four stages, corresponding roughly to conventional software engineering practice. In the first three stages students were required to submit written reports covering the project requirements, user-interface design and then detailed design of their proposed program. At each stage, groups would exchange reports and write a summary and critique of another group's project. Instructors also provided detailed written feedback. Before seeing the written critiques, each group had to present their report orally to the whole class and receive feedback and criticism from everyone. Reports were then rewritten based on the feedback. Each of these activities was graded on a group basis, including the comments/questions in the classroom. The fourth stage was implementation, culminating in an open demonstration of the, hopefully functional, software. The source code was checked and an overall grade was assigned for this stage too. The reports accounted for 25%, critiques and presentations for another 25%, and the implementation and coding for only 15%, reflecting the emphasis on design, communication and teamwork. While students were certainly encouraged to complete the implementation of their projects, it was recognized that time was limited and clearly indicated that demonstrating a finished working program was not essential.[3] Providing a solid foundation had been laid (in terms of design and coding) so that they or others could complete the project (without having to throw everything away and start again), then this was perfectly acceptable. In order to encourage cooperation, a local Unix newsgroup was set up to serve as a forum where students could post problems related to their implementation and hopefully receive answers from other groups. Additional grades were awarded for publishing helpful bits and pieces. The final component of the project mark was a peer grade given secretly by fellow group members and accounting for 5%.

A course webpage [10] provided a convenient means for coordination of the course, ensuring that all instructors, assistants and students, knew what was required of them each week. It also provided a mechanism for distributing lab assignments and course handouts. Students were expected to check the webpage frequently and were also encouraged to submit queries to the instructors and assistants via email. By providing up-to-date information and quick responses it was hoped to get students to use and appreciate this new technology.

VI. First Impressions

With objectives clear and the means of achieving them planned out, the semester began with more than 100 students in two sections. Students were told what was expected of them. It was also explained to them that this was something of an experiment, in that, it was the first time such a course was being attempted and things might not work out exactly as planned. Indeed, there were difficulties, but mostly minor. Initial presentations overran badly, so much so that written critiques of the second and third reports had to be dropped in order to stay roughly on schedule. Plans to arrange a sort of "open day" where students could show off their projects before the semester finally ended, also had to be dropped.

There were technical difficulties too, related both to installation and programming. For example, although students could read the newsgroup, posting to it proved to be very difficult. This was eventually resolved by getting students to send email to the instructor, who then posted the message on the group's behalf. There were also difficulties with the particular network configuration of Delphi, especially the database engine, which resulted in considerable delay and frustration for some groups. On the programming side too, problems sometimes took a long time to solve, due mainly to a lack of familiarity with Delphi and the fact that no specialist books were then available in the library. Ignorance in this regard did have one unexpected benefit, however, in that problem solving became a joint activity. The resulting atmosphere, in which both students and staff tried to resolve problems together, was particularly rewarding.

Projects undertaken included several advanced versions of Robo (using color, animation, and 3-D), a calculus tutor, a digital circuit simulator, a bus seat reservation system, a patient tracking system, a join the dots game, a geography quiz, and a paint program. They were all potentially marketable products, most employing not just a GUI, but also online help and even databases.

VII. Survey Results

To evaluate the new style course, an anonymous survey was conducted at the end of the semester. The results, presented below, seem very encouraging given that this was the first time a project-based approach had been tried.

[3]Lack of time also meant that students would have little chance to experience the testing and maintenance phases of the project. In part, this issue was addressed by organizing a project open day/contest the following semester. It was hoped that students would complete their work and demonstrate it there; public recognition and possibly cash prizes providing the incentive!

Overall, there was general understanding that the course had proved valuable. Some 60% of the students said that they began the university with absolutely no prior programming experience. Most viewed CS101 and CS102 as a well-integrated introduction to computers and programming. Students were given the choice of doing their projects in Delphi under Windows or in Pascal under MSDOS. More than 70% of the class opted for Delphi. There were a few complaints about the failure to explicitly teach Delphi and it appears that devoting a few (more) class hours to this may indeed have been a good idea. There was almost unanimous agreement amongst students that they should gain experience with sophisticated tools like Delphi, although about 30% suggested it might be better in the following year! Around 35–40% of students who undertook a project in Delphi found it quite difficult and suggested that maybe it was too big a jump. On the other hand, students were not expected to become proficient in Delphi (recall that only 15% of the grade was allocated for implementation). Furthermore, the Delphi projects were generally better and the groups seemingly more motivated than those doing Pascal projects, possibly due in part to the relative ease of producing sophisticated, professional looking programs using such a tool. It is pleasing to be able to report that when students were asked to state what they thought were the major objectives of the course, they did so reasonably accurately. Perhaps more importantly, when asked what major skills or lessons they felt they had learned in the course, their responses showed considerable overlap with those objectives (essentially only the ordering was different).

More than 60% of students reported difficulties working together in a group. The major complaints were members not working, and the problem of gathering and communicating. In fact, lab hours were often left free so that groups could work together, however, they were not required to do so. It seems that many did not take advantage of such times and, moreover, failed to find other times to compensate. Generally, six people groups were considered ok, but many thought it would be easier/better with fewer people, the ideal being 4.5! One group completely fell apart, unable to meet or agree upon anything. It was eventually accepted that its members would complete the project individually. Ultimately, only two of them did any work at all, the others simply dropping out despite attempts to persuade them they could succeed.

Students estimated that they spent an average of six to seven additional hours per week on CS102, more than on other first-year courses. Many felt that there was too much emphasis on the project and suggested more weight be given to conventional lectures and assignments. Electronic communications proved very successful, even the newsgroup despite suffering from the technical problems mentioned earlier. Although there were problems with computers, including technical difficulties, actually getting access to a machine at certain times, and working in a crowded noisy laboratory environment, more than 60% "completed" their projects. All reported being very pleased/proud of the work they had done.

Last, but by no means least, only 20% viewed report writing as a waste of time. Some 40% said they found it valuable, while the other 40% saw it as quite valuable. This is, perhaps, one of the major achievements of the new style course. Never before had students been convinced that putting their design ideas on paper was a worthwhile, even essential, part of programming. Finally, it seems, the message has gotten across.

VIII. Concluding Remarks

The survey results seem to indicate that students have understood and appreciated the essentials of engineering design, indicating that the course has successfully fulfilled its objectives. Consequently, it will be repeated in the same form next year. Some minor changes are begin considered, such as requiring students to meet in the scheduled lab periods and assigning assistants as observers to try to ensure more equality in work load. In response to student pressure it is planned to change the project/exam ratio slightly to 60/40, however, the group size will remain the same both to ensure a manageable number of groups and to make sure that groupwork problems are likely to arise!

Reading reports, actively listening to and criticizing presentations, and finally helping resolve implementation problems, demanded a lot of time, not to mention patience, on the part of the instructors. Despite this, however, they strongly believed that it was all worthwhile and certainly more enjoyable than reading hundreds of identical examination papers! Major differences in exam scores or in performance on later courses (except possibly those related to design) are not expected, given that the objectives are generally different. While the really important benefits are perhaps impossible to quantify, the immediate outcome is clear. The amount of effort students expended on their projects, and the sheer joy and pride they had in their work were obvious. They were active and they were involved. They had learned something!

Whether or not it is really necessary to teach programming in first-year undergraduate engineering classes, the restyled course described here would appear to offer significant advantages. In the interim, it provides a vehicle whereby those students who have not yet been exposed to the computer, can be brought up to speed. More importantly, focusing on those skills which are central to the practice of engineering (methodology, communication, teamwork, and independent learning and motivation) and using the computer merely as a means to develop them, helps justify the inclusion of such a course in the curriculum of both CS and non-CS majors.

1) Postscript: As the new semester approaches, the dreaded question of language once again raises its ugly head. This time the suggestion is to switch to Java. Although it retains many of C's idiosyncrasies, Java is a relatively clean object-oriented programming language. Being not only cross platform (Unix, PC, and Mac compatible), but also Internet compatible, its popularity is assured and visual development environments similar to Delphi are already appearing, such that switching the course to Java would not prove too difficult. The major problem, however, is that Java is still very new, and subject to substantial and rapid change. Compared to Delphi, it is also quite restricted and pedagogically not as nice, requiring students to include a substantial amount of initially "incomprehensible" code for even the most basic of programs. Early adoption of Java is thus a somewhat daunting prospect, especially considering that there are only a few introductory-level textbooks presently available.

ACKNOWLEDGMENT

The author would like to thank his co-instructor, Dr. E. Tyn, the CS101/2 assistants and, of course, all the first-year CS and EE students at Bilkent, who worked so hard to make this experiment successful. A special thanks is also due to O. Özhan, for collating and analyzing the survey results, and to Prof. M. Baray, Dean and Head of Department, for allowing us the flexibility to implement this program.

REFERENCES

[1] W. K. Durfee, "Engineering education gets real," *Technol. Rev.*, vol. 97, no. 2, p. 42, Feb/Mar 1994.

[2] E. B. Koffman, *Turbo Pascal*, 5th ed. Reading, MA: Addison-Wesley, 1995.

[3] [online] Available http://www.acm.org/education/curr91/homepage.html A. B. Tucker, Ed., *Computing Curricula 1991, Report of the ACM/IEEE-CS Joint Curriculum Task Force,*: ACM Press and IEEE Computer Society Press, 1991.

[4] D. Davenport, "Robo: A programming system for teaching introductory software engineering concepts," in *Proc. ISCIS-V Int. Symp. Comput. Inform. Syst.*, Cappadocia, Turkey, Oct. 30—Nov. 2 1990, pp. 953–962.

[5] S. Papert, *Mindstorms: Children Computers & Powerful Ideas*. New York: Basic Books, 1980.

[6] N. J. Yelland, "Encouraging young children's thinking skills with Logo," *Childhood Educ.*, vol. 71, no. 3, p. 152, Spring 1995.

[7] Robo webpage. [Online] Available http://www.cs.bilkent.edu.tr/~david/robo.htm

[8] L. F. Johnson, "C in the first course considered harmful," *Commun. ACM*, vol. 38, no. 5, p. 99, 1995.

[9] Delphi. Turbo and Borland Pascal are products of Borland Int. [Online]. Available http://www.borland.com

[10] CS102 course webpage. [Online]. Available http://www.cs.bilkent.edu.tr/~david/cs102

David Davenport received the B.Sc. and Ph.D. degrees in electrical and electronic engineering from the University of Birmingham, U.K.

Previously he was Electronics Design Engineer, Independent Software Consultant, and Assistant Professor of Marine Sciences. He is currently an Assistant Professor in the Computer Engineering and Information Science Department, Bilkent University, Ankara, Turkey, where he acts as Lecturer and Coordinator for all first-year undergraduate programming classes in the engineering faculty. His research interests now center on cognition and the use of computers for education.

Dr. Davenport is a Member of the ACM and Cofounder and Acting Chairperson of the local ACM SIGART group.